Description of Projects Utilizing Equipment Purchased under the DURIP Award

Introduction

Intrusion detection and, more recently, automated response to intrusions have been the cornerstones of our research programs in the Computer Security Laboratory at UC Davis. UC Davis has pioneered several novel methods for intrusion detection on large networks. This research, however, can progress only if the laboratory's network infrastructure is sufficiently large to realistically model that of a large organization. UC Davis is also among the first to study cooperative automated response to intrusions. Network devices: routers and firewalls, exchange attack information which can be used to optimally filter the malicious traffic. This project depends heavily on having network hardware on which to install and test our prototypes. Thanks to the generous DURIP equipment grant, our group was able to obtain the resources necessary for this work to proceed.

Intrusion Detection

GrIDS

The GrIDS system[25], developed at UCD under contract from DARPA, is designed to detect large-scale network activity such as extensive network probes (doorknob rattling), worm attacks, coordinated attacks from several sites, in addition to network policy violations. GrIDS can utilize reports from host-based IDSs, network sniffers and network tracing and accountability tools, to build pictures of correlated network activity as graphs. In addition to giving a network context to attacks which can actually be detected at a single place, GrIDS is capable of detecting attacks which are not visible at any single place but show up on a large scale as anomalous patterns of activity.

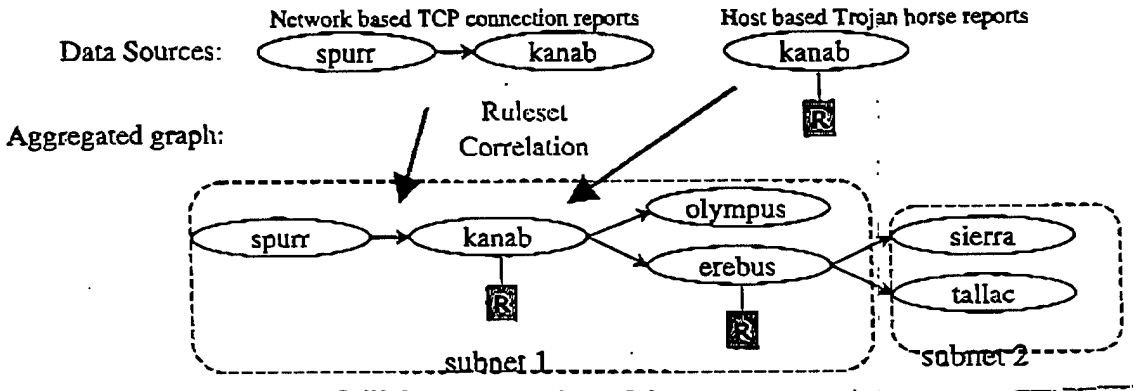


Figure 1: GrIDS representation of the worm scenario

REPORT DOCUMENTATION PAGE

AFRL-SR-BL-TR-00-

0293

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	Final Technical Report 01 Mar 98 to 28 Feb 99
	22 Jun 00	

4. TITLE AND SUBTITLE
Description of Projects Utilizing Equipment Purchased under the DURIP Award

5. FUNDING NUMBERS
F4960-98-1-0269

6. AUTHOR(S)
Professor Karl N. Levitt

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of California, Davis
Department of Computer Science
1050 Engineering II, One Shield Avenue
Davis, California 95616-5294

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
801 N. Randolph St, Rm 732
Arlington, VA 22203-1977

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-98-1-0269

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Intrusion detection and, more recently, automated response to intrusions have been the cornerstones of our research programs in the Computer Security Laboratory at UC Davis. UC Davis has pioneered several novel methods for intrusion detection on large networks. This research, however, can progress only if the laboratory's network infrastructure is sufficiently large to realistically model that of a large organization. ITC Davis is also among the first to study cooperative automated response to intrusions. Network devices: routers and firewalls, exchange attack information which can be used to optimally filter the malicious traffic. This project depends heavily on having network hardware on which to install and test our prototypes. Thanks to the generous DURIP equipment grant, our group was able to obtain the resources necessary for this work to proceed.

14. SUBJECT TERMS

15. NUMBER OF PAGES
14

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

A GrIDS graph consists of nodes representing hosts or routers, and edges representing network traffic between two nodes. Graphs are built by aggregating together activity (either at nodes or on edges), which seem to be correlated. Correlation is flexibly defined, but typically includes proximity of the events in time, and may also include other measures of similarity (e.g. whether two TCP connections have the same destination port). Each node and edge is decorated with information providing more detail about the connection. Typical edge decorations concern type of connection, duration of the connection, anomalous aspects of the connection, and the user if known. Node decorations contain information such as attack reports, host OS type, and any other information which may be passed up to the GrIDS network-wide system. The graphs are the result of analysis of incoming reports according to rules created by the system security officer. (Simple default rules are provided). These rules describe how graphs are created, modified and removed, and which activity is considered dangerous or in violation of system-wide access policies. GrIDS can represent a single connection in many different types of graph, thereby viewing connections in different ways.
GrIDS is a hierarchically-controlled system, in that GrIDS components communicate activity in sub-domains to components at a higher level of abstraction; the components at the higher level can then make inferences about activity occurring across a larger portion of the network. These components can then direct components at lower levels to act or communicate warnings to hosts in the sub-domains. GrIDS components communicate with each other using reduced representations of graphs which cross domain boundaries; a component receiving one of these reduced graphs can request the full graph should it need more information.

Commercial IDS Evaluation

Intrusion detection software is now available from several commercial vendors. There is a difficulty, however, in interpreting the true capabilities of these products from their marketing claims. Each claims to detect similar types of computer attacks but it isn't clear from the documentation how their algorithms differ or whether they rely upon the same attack signatures. We are engaged in an independent evaluation of these products that compares the behaviors of different commercial systems when observing identical attacks. Oftentimes the IDS is a network appliance and, rather than running as an application on top of an independently installed operating system, the IDS runs on as the system on a dedicated server. This requires an entire standalone machine.

Watchers

The goal of WATCHERS is to identify bad routers, which are routers that drop or misroute packets, and routers that do not participate correctly in the WATCHERS protocol.

WATCHERS Counters

Each router counts every packet that arrives or departs over its interfaces. Each router also separately counts arriving packets that have been misrouted.

Data Byte Counters

Henceforth we will refer to the first router in a packet's route as the packet's source, and to the last router in a packet's route as the packet's destination. For any pair of routers, we identify three types of packets for each direction of traffic. Each router counts the number of data bytes in each packet, using a separate counter for each type. (WATCHERS counts data bytes instead of entire packets because packets are sometimes legitimately fragmented into smaller packets by routers. Thus, the number of packets that go into a router per unit time does not necessarily equal the number of packets that come out.) Figure 4 illustrates the three types of packets and the associated counters. Consider traffic flowing from X to Y. The $T_{X,Y}$ counter refers to packets that pass through both X and Y. The $S_{X,Y}$ counter refers to packets with source X that pass through Y. The $D_{X,Y}$ counter refers to packets with destination Y that pass through X. The other three counters are similar, but for the opposite direction of flow. Routers X and Y both maintain copies of all six counters to check each other during the WATCHERS diagnosis phase.

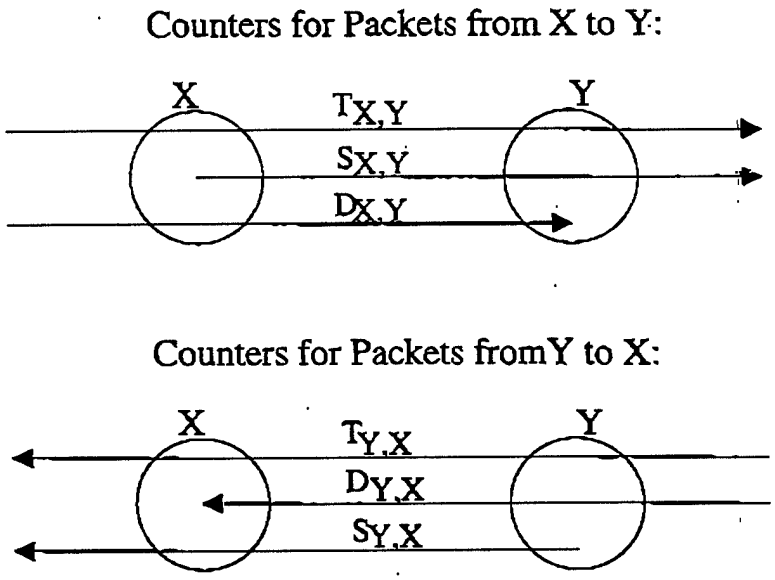


Figure 4: Packet byte counters

Misrouted Packet Counters

Each router also counts the number of times each of its neighbors misroutes traffic. For each neighbor X of router Y, the counter $M_{X,Y}$ records the number of times router X wrongly, with respect to the optimal path, forwards (*misroutes*) a packet to Y. To detect such misrouting, each router maintains a copy of each neighbor's routing table. If an M-counter value is larger than an acceptable threshold (possibly zero), then WATCHERS identifies the associated neighbor as a bad router during the WATCHERS diagnosis phase.

Communication in WATCHERS

Each router executes the WATCHERS protocol at regular intervals; each execution of WATCHERS is a round. In each round, each participating router in the AS runs the protocol concurrently. The first part of each round is devoted to communication. Each

router records a "snapshot" of its counter values, and sends a message containing the values to all of the other routers. When a router has received all of the counter values it needs, it begins the diagnosis phase of WATCHERS.

Diagnosis

The WATCHERS diagnosis algorithm has three parts: *preliminary checking*, *validation*, and *conservation-of-flow analysis*.

Terminology and Concepts

A *testing router* is a router running the diagnosis algorithm to test its neighbors (the *tested routers*). A *shared link* is the link between a testing router and a tested router. To test another router, a testing router needs to use its own counters, the tested router's counters, and the counters from each of the tested router's neighbors.

Preliminary Checking

First, the testing router checks if any neighbor has misrouted too many packets, or violated any part of the WATCHERS protocol. Each such neighbor is diagnosed immediately as bad.

Validation

During validation, a testing router examines its neighbors' counters. Each neighbor's counters for the shared link should match the testing router's corresponding counters. A discrepancy between any pair of corresponding counters indicates that the tested router is bad. (Small discrepancies can be tolerated, as explained later.)

Each testing router must also check if each neighbor's counters match the counters of *their* neighbors. If a neighbor n has a discrepancy with one of its neighbors t, then the testing router does not perform the next stage of diagnosis (conservation-of-flow analysis) on n. Instead, the testing router assumes that n will diagnose t as bad, or vice versa.

Validation cannot locate all bad routers. Validation identifies routers that drop packets and then change their counters to conceal their behavior. To find bad routers that drop packets but leave their counters intact, we use conservation-of-flow analysis.

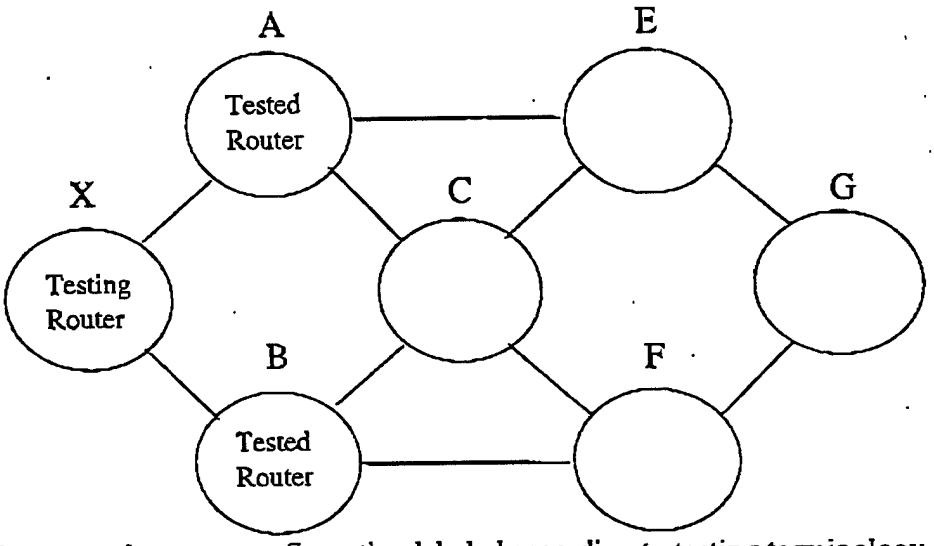
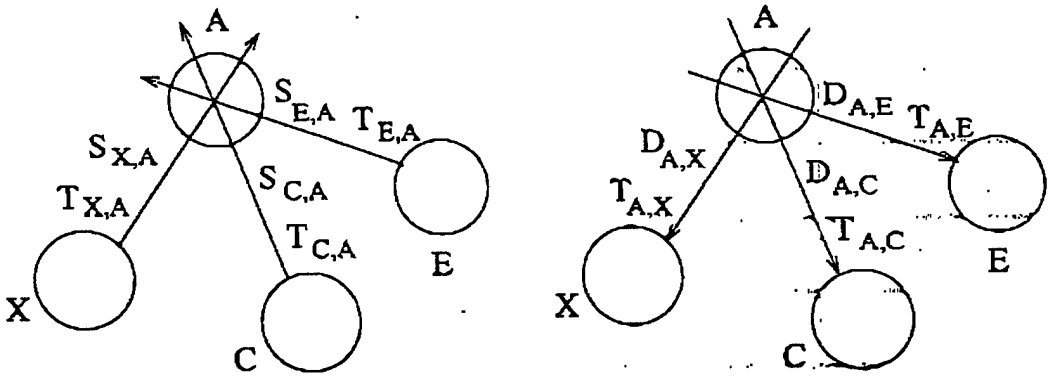


Figure 2: A sample router configuration labeled according to testing terminology.



Incoming Traffic Flow Outgoing Traffic Flow

Figure 3: Incoming Traffic flow vs. outgoing traffic flow. X testing A from Figure 2.

Conservation-of-Flow Analysis

In the second stage of diagnosis, a testing router performs conservation-of-flow analysis on each neighbor.

For example, if router *X* in Figure 2 is the *testing router*, it performs a test on routers *A* and *B*, its neighbors. To test the flow through *A*, *X* needs counters from *A* and each of *A*'s neighbors: *C*, *E*, and *X* itself. To test the flow through *B*, *X* needs counters from *B*, *C*, F, and *X* itself.

Conservation-of-flow analysis is based on a simple principle: in any time interval, the number of data bytes going into a router (less the number of bytes destined for the router) should match the number of data bytes that come out of the router (less the number of bytes sourced by the router). The former quantity is the *incoming traffic flow* and the latter quantity is the *outgoing traffic flow*.

For example, suppose *X*, in Figure 2, wants to test *A*. *X* can calculate *A*'s incoming traffic flow by using the counters for *A*'s incoming edges (i.e., the counters tracking flow from routers *X*, *C*, and *E*). The expression for *A*'s incoming traffic flow (I_A) is:

$$I_A = \sum_{\forall N | A \leftrightarrow N} (S_{N,A} + T_{N,A})$$

(where the notation $A \leftrightarrow N$ indicates that A and N are neighbors.) The summation does not include $D_{N,A}$, since this counter is for data bytes destined for *A*.

Similarly, the expression for *A*'s outgoing traffic flow (O_A) is:

$$O_A = \sum_{\forall N | A \leftrightarrow N} (D_{A,N} + T_{A,N})$$

The counters needed to compute the incoming traffic flow and the outgoing traffic flow are illustrated in Figure 3. Now, to test *A* for conservation of flow, *X* simply compares *A*'s incoming traffic flow to its outgoing traffic flow. The difference in the two values should be less than the *fault tolerance threshold.*

if (I_A - O_A) > threshold **then** *X* diagnoses *A* as a network sink;
if (O_A - I_A) > *threshold* **then** *X* diagnoses *A* as bad
(because *A* appears to be artificially generating packets).

The threshold value can be set to 0 for no fault tolerance. We discuss thresholds further later.

Response

During a round of WATCHERS, a testing router can diagnose a neighbor as a bad router for a number of reasons. In all cases, the testing router takes the same actions. It floods a routing update advertising the shared link as down (inoperable), which signals the other routers in the AS to remove that link from their network maps. Also, the testing router ceases to send packets along that link and acknowledge traffic received on that link. After all of a bad router's neighbors take these steps, the bad router is *logically disconnected* from the network.

Critical Research Issues

To determine if it is practical to use WATCHERS in real networks, we are investigating some critical issues. We describe two issues here: WATCHERS' scalability and fault tolerance.

WATCHERS' Scalability

WATCHERS would be most valuable running in a large autonomous system, where it is difficult to manually check and maintain the security of each router. Thus, WATCHERS' scalability is a significant issue. Specifically, are studying WATCHERS' memory, communication, and processing costs, and how these costs increase as the AS becomes larger. We have demonstrated that these that these costs depend heavily on the number of routers in the AS and the number of edges in the graph that models the AS (the *router graph*). In addition, the communication and processing costs are proportional to the

frequency of WATCHERS rounds. Table 1 shows the worst-case costs for both a ``full AS," in which each router has a link to every other router, and for a ``sparse AS," which we define to be an AS in which each router has a maximum of three neighbors. We analyzed costs in a full AS to show the ``absolute'' worst case, and we analyzed costs in a sparse AS because it is more realistic.

Resource	Units	Cost: Sparse AS	Cost: Full AS
Memory	bits/router	$O(R^2)$	$O(R^3)$
Processing	CPU cycles	$O(R^2)$	$O(R^4)$
Communication	bits/AS per WATCHERS round	$O(R^3)$	$O(R^5)$

Table 1: WATCHERS' Costs

Our analysis illustrates that WATCHERS' memory, communication, and processing costs can be expensive for a full AS but reasonable for a sparse AS. In both cases, whether or not WATCHERS is practical depends on the frequency of WATCHERS rounds.
In our analysis, we did not include the costs of a message authentication system, which would have added costs in all three categories.

Fault-Tolerance Thresholds

Ideally, the data byte counters of neighboring good routers should match perfectly, and for each good router the corresponding misrouting counter values should be zero. In practice, though, the counter values may be different than the ideal values, even for good routers. Three reasons why counter values may be different from the ideal or expected values follow.

- Typical network protocols, including several within the TCP/IP suite, drop packets due to errors in transmission or congestion in the network [6]. Thus routers with extremely heavy traffic may drop packets at significant rates.
- Neighbors may not be perfectly synchronized when a new round of WATCHERS begins due to propagation delays for the request messages. Assume that there are N routers in the AS, and let $k = ceil(N/2)$. According to the RRR sub-protocol, each router records a snapshot of its counter values as soon as the kth request messages arrives. We call this instant the *recording time*. The difference in recording times for two neighbors should be less than or equal to the delay on their shared link, since flooding is used to distribute request messages. However, any difference in recording times may cause the counter values of neighbors to disagree.
- Our misrouting detection method depends on neighboring good routers to agree on the network topology. Since a good router sends routing updates whenever it changes its own routing table, and its good neighbors immediately change their own tables accordingly, any period of discrepancy between the routing tables of neighboring good routers should be brief [8]. Moreover, in modern link-state routing protocols, link costs are static and therefore link-state changes are infrequent [13]. Even so, a momentary disagreement over topology may cause a router to increment an M counter with the mistaken belief that a neighbor has misrouted a packet.

Therefore, WATCHERS should accept a counter value that is different from the ideal value if the difference is less than a pre-selected threshold.

The threshold values will depend on the WATCHERS *period* (time between consecutive rounds). The period should be adjusted to fit the environment. The period must be at least as long as the time needed by the slowest router to execute one round of WATCHERS. The period should be short enough to avoid counter overflow. Since the threshold values depend on the period, the period and thresholds should be adjusted together.
Setting the thresholds correctly is critical. If the threshold values are too small, too many false alarms may occur. If they are too large, too many attacks might escape detection. However, in some environments, choosing thresholds may be difficult, due to rapidly changing conditions in the network (e.g., network load). Such environments may require threshold values that change dynamically.

Automated Response to Intrusions

With current intrusion detection technology, it is possible to detect attacks in real time. Typically, when an IDS system is triggered, a human operator is notified. The system is then adjusted manually in response to the intrusion. Many types of attacks, however, can only be thwarted if the response is quick. Responding quickly, without considering the self-inflicted damage that a response might inadvertently cause, however, is equally dangerous. Should the attacker discover the response mechanism, a willful triggering could be a denial-of-service attack in itself.
To date, response systems have employed a multi-layered approach to the problem. A locally controlled response layer, combining knowledge collected locally with information sent from its immediate neighbors, is responsible for quick actions not requiring time-consuming calculations and can stop attacks in progress. These responses are necessarily severe and thus are deliberately limited to be short-lived to avoid massive denial of service.
Clearly, a more intelligent response; a response that minimizes the self-inflicted damage, is possible if more time is allowed for the optimal solutions to be generated. This is implemented as a response layer superimposed atop the local components. Local response agents, in addition to sending messages to their neighbors, will send attack and response information to a central processor for aggregation, response selection and damage minimization. Damage estimates are derived from a policy that attempts to capture the value of important system services in a parameterized model.
The Intrusion Detection and Isolation Protocol (IDIP) developed by a collaboration of UCDavis, Boeing and NAI labs, represents a first attempt to produce this type of cooperative response system. IDIP is a protocol whereby filtering routers, firewalls and host-based response modules cooperate with intrusion detection systems (IDS) to trace the attack back to the source and stop it. An IDIP enabled network includes a centralized Discovery Coordinator (DC). The IDIP DC operates on a global level, incorporating information from multiple IDS reports and IDIP enabled sources. It contains a map of network topology and a list of major services with their relative values. It recognizes coordinated attack patterns, determines the optimal response and the placement in the network that minimizes the impact upon critical services. If the least costly response is worse than the potential damage due to the attack, no response is issued.
Considerable effort has been undertaken to produce an IDIP DC that optimizes responses from the point of view of the entire protected network. Although the Discovery

Coordinator is in the best position to calculate the optimal response tactic; oftentimes the time required to arrive at the solution makes issuance of the response irrelevant. If a site is attacked by a self-propagating worm, for example, significant time is required to recognize the traffic pattern, to decide how and where to respond and to issue the response directive to the appropriate boundary controllers. The local IDIP components, on the other hand, have direct access to the traffic logs containing the record of packets used in worm propagation. Involving the local IDIP components in the response strategy, then, insures that the system can react in the most timely and effective manner. Additionally, the Discovery Coordinator represents a single point of failure, the loss of which results in the total loss of the ability to optimize responses. Adding the capability to optimize the global response posture to the local IDIP response component results in a more robust fault-tolerant response system that continues to operate even with the loss of several components. Previously, our method for response selection was based upon ad-hoc, plausibility arguments supported by scenario based case studies. We are now moving beyond this to a more formal model of response. Specifying response strategies and behaviors using a state machine description allows us to analyze our responses for arbitrary network topologies and detector locations and to prove that our responses will be valid for any configuration.

Globally Optimized Response Using Local Control

Modeling Response – A State Machine Approach

First attempts at tackling automated response have relied upon ad-hoc response heuristics. The IDIP protocol, for example, implements at network boundary devices the strategy "Upon receiving an attack report, check whether relevant attack traffic was passed. If so, filter the traffic and forward the attack report to immediate neighbors". The goal for this strategy is twofold: to block the attack before it completes, and to take the blocking action as close as possible to the attacker's location. Evaluation of this as an effective strategy was done heuristically; various attack scenarios and a variety of network topologies were considered to verify that the correct response was obtained in each case. Implementing new strategies, then, proceeds in an ad-hoc manner with the heuristic process repeated for each one. As more complex response strategies are considered, it becomes unlikely that all pathological network configurations have been taken into account. With certain topologies, for example, there may be alternate paths for neighbor messages to propagate causing infinite IDIP message loops. Lack of a model strictly defining response component behavior has resulted in the implementation of only simplistic locally controlled responses whose aggregate, network-wide response is overly severe.

We are currently developing a state-machine model of local response agent behavior. Nodes in the state diagram represent response postures of an enabled device. State diagram edges represent transitions that occur when messages are received from neighboring devices and specify messages sent to neighboring devices when making the transition. Including more sophisticated response strategies proceed by adding additional neighbor-to-neighbor messages (edges) and intermediate device states (filtering postures). Local response strategies that are designed to minimize the negative impact on normal system operation can then be built up from simpler strategies already developed

and proven. Rather than using case studies to argue that responses are reasonable, the state machine description provides a specification against which proofs can be employed verifying their validity when applied to arbitrary network topologies and detector locations.

The Current IDIP IDIP Implementation in the State Machine Model

As an example of the state-machine model, figure 1 shows how current IDIP locally controlled response would be specified.

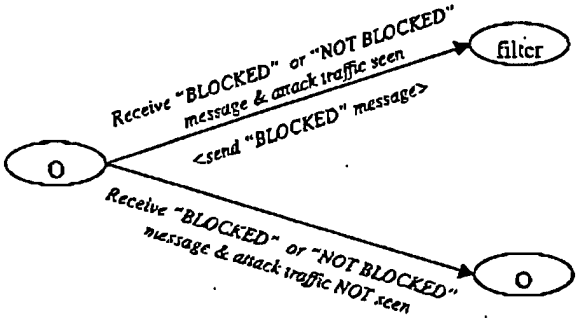


Figure 1: State-machine specification of the current local IDIP response. Devices that have seen the specified attack traffic will filter to block the attack and inform their neighbors with a "BLOCKED" message. Devices that haven't seen the traffic will do nothing.

The aggregate result of this simple response strategy, when each local agent implements it independently is all response-enabled devices along the attack path filtering identical traffic. Clearly this is an overly severe response. Only the device closest to the attacker need filter traffic. After it implements its response, the filtering on all the downstream devices serves only to deny legitimate service and are clearly redundant. Current solutions rely upon a centralized network level process to recognize this and direct the downstream devices to remove their redundant filters.

State-Machine Extensions to Remove Redundant Responses

We take advantage of the state-machine model by adding additional message types and response state nodes to produce a more complex and effective response strategies. Then, when an attack is detected, the initiating device can request from among a collection of response strategies when informing its neighbors. Local devices then gain response capabilities previously reserved for the global response component. A local response agent knows the direction of the attack traffic that it has forwarded. If it receives a "BLOCKED" message on the same interface as the source of the attack packet(s) it knows that an upstream device has responded. Any further filtering, then, will be redundant. Making simple extensions to the state diagram that specifies current IDIP behavior implements this new response strategy that reduces the impact of aggregate behavior on global network operations. Figure 2 shows the state diagram specifying the new behavior.

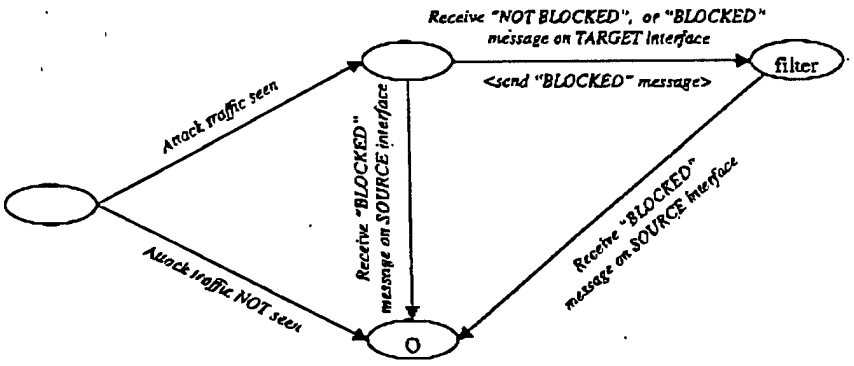


Figure 2: State-machine specification of a local response strategy that eliminates redundant responses on the global network. Devices remove any filtering when a closer device has responded.

Application of this response strategy locally results in a reduction in the impact on network resources when seen from the global perspective. Rather than having multiple identical and redundant responses to an attack, a single filtering router blocks the malicious traffic and the downstream devices are free to route normal system traffic unimpeded. In addition, relying only upon the interaction of local components to produce a globally optimized response results in a more robust, fault tolerant system.

Extensions Producing Non-Localized Responses

The state-machine approach also provides a natural procedure that can be used to obtain responses even in components not directly affected by attack activity, a task typically performed by the global, central response processor. Many network topologies contain multiple alternate paths between source and destination machines. When one link becomes overloaded or unavailable, traffic can still continue over the alternate path(s). When response actions that filter traffic are successful in isolating the attacker, packets could still be routed via an alternate link. Responses then would always remain one step behind the attacker, with normal network fail-safe procedures providing access to the target machine.

Extensions to the state-machine model can also produce a global response posture that includes machines that haven't yet seen any of the attack traffic. Figure 3 shows a state-machine specification of the attack strategy that will isolate the attacker on *all possible* attack paths, not just paths previously taken by the attacker.

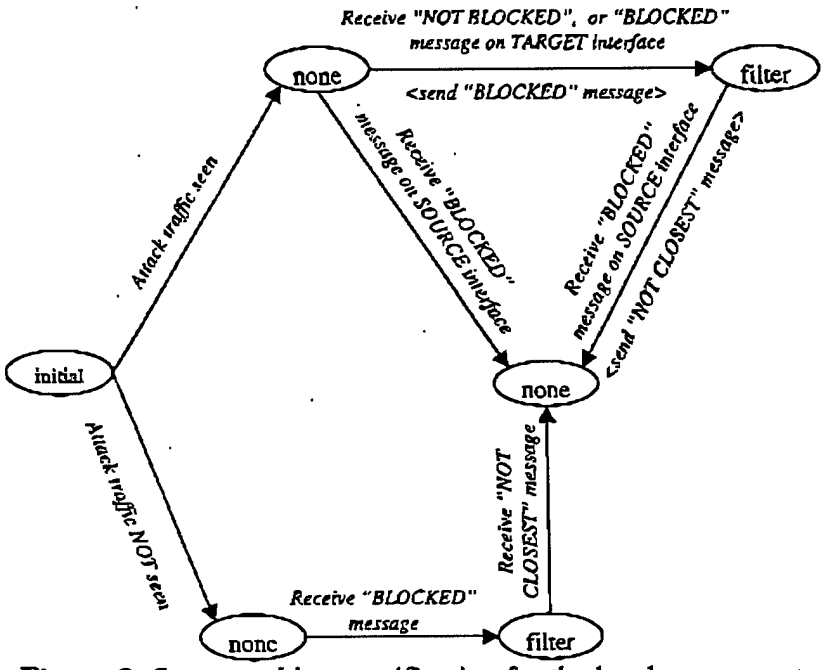


Figure 3: State-machine specification for the local response strategy that will result in the aggregate global response of "Block along all possible paths".

This new strategy is a simple extension of the strategy presented in the previous section. A new state that produces a traffic filter response in devices that haven't seen any of the malicious traffic has been incorporated. These devices now function as "helpers"; when their neighbor blocks the attack they will also block, isolating the entire neighborhood. This alone, however, is insufficient since there is no mechanism for these filtering devices to back off their response once the device they are "helping" realizes it is not the closest. Thus a new "NOT CLOSEST" message is added that tells these "helper" devices to remove their filtering actions.

Further extensions

In this manner, a rich set of global filtering postures is generated. In addition to the strategies described above, we have also developed a state-machine specification for a local response that isolates a detected self-propagating worm. Table x shows the messages required for the strategies described above. We envision extending the response strategies to include cost to mission critical resources. Analogous to the "NOT CLOSEST" message, we might incorporate a "NOT CHEAPEST" neighbor-to-neighbor message that allows the response to be placed at the location with least cost to critical mission resources. We also propose to incorporate trust levels into our response strategies. A particular local device may choose not to filter based on messages from neighbors that it doesn't trust completely, replying with a "NOT BLOCKED" message to the untrusted device.

An Intrusion Detection and Response Test Bed

Pursuit of the previously described research projects requires a heterogeneous collection of computers to host the intrusion detection systems, and a number of machines to serve

as network infrastructure devices, such as routers and firewalls. The DURIP grant awarded to the UC Davis Computer Security Lab has been instrumental in enabling the purchase of the necessary hardware.

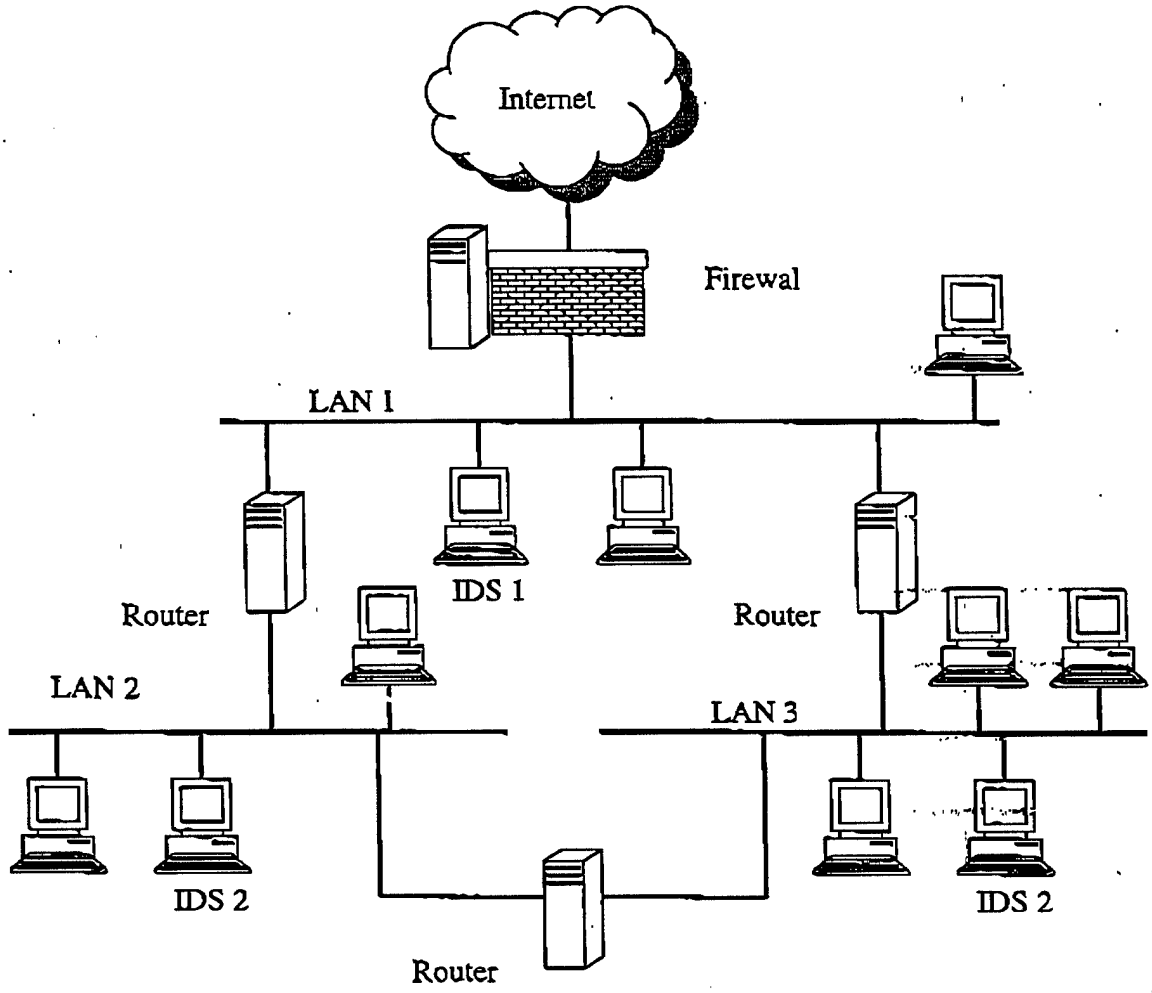


Figure 0: Network topology of test-bed created from DURIP grant for intrusion detection and response research projects.

To prototype our response systems and the watchers protocol, we require access to the source code of the network devices. To this end we purchased a large collection of Intel (I386) based machines. We configured several of these machines with multiple network interface cards to create routers and firewalls for our ID&R research testbed. Our testbed is composed of a single firewall offering a single interface to the Internet. Internally we used and additional three machines to serve as routers. These are configured, as shown in Figure 1, to create three separate LAN segments in a topology containing a possible routing loop. On the three LANs are one or more machines running various IDS systems. We use this to correlate the output of multiple systems reacting to the same attack observed at different locations in the network. In addition, by running open-source operating systems on the routers and firewalls, we have the freedom to insert our

prototype intrusion detection and response systems into the underlying network servers. Intrusions detected at one point in the network inform the customized routers and firewalls which then implement a response action.